

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 04-163625

(43)Date of publication of application : 09.06.1992

(51)Int.Cl.

G06F 9/06

(21)Application number : 02-291013

(71)Applicant : FUJI XEROX CO LTD

(22)Date of filing : 29.10.1990

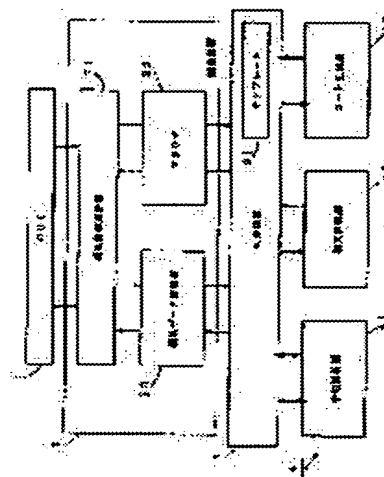
(72)Inventor : IWATA MASATAKE
KURABE ATSUSHI
ICHIJI HIROSHI

(54) SOURCE CODE GENERATION DEVICE

(57)Abstract:

PURPOSE: To control information with much versatility by absorbing the difference of the output format of a source code based on the difference of GUI construction surroundings by means of a template.

CONSTITUTION: A system consists of GUI (graphic user interface) 1, an editing device 2, a storage device 3 and a source code generation device 4. The template 31 where a character string which is directly added as the source code being the object of generation and information which requires correction to a content whose construction is analyzed for making the source code being the object of generation are described is provided. Different code generation elements between different GUI construction surroundings are absorbed through the template 31. Thus, the source code of the target program can conventionally be generated for various GUI construction surroundings.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

⑩ 日本国特許庁(JP)

⑪ 特許出願公開

⑫ 公開特許公報(A) 平4-163625

⑮ Int. Cl.⁵

G 06 F 9/06

識別記号

4 3 0 E
4 3 0 J
4 3 0 M

庁内整理番号

7927-5B
7927-5B
7927-5B

⑭ 公開 平成4年(1992)6月9日

審査請求 未請求 請求項の数 3 (全18頁)

⑬ 発明の名称 ソースコード生成装置

⑯ 特 願 平2-291013

⑰ 出 願 平2(1990)10月29日

⑱ 発 明 者 岩 田 正 武

神奈川県川崎市高津区坂戸100番1号 KSPR&Dビ
ネスパークビル 富士ゼロックス株式会社内

⑲ 発 明 者 倉 部 淳

神奈川県川崎市高津区坂戸100番1号 KSPR&Dビ
ネスパークビル 富士ゼロックス株式会社内

⑳ 発 明 者 伊 知 地 宏

神奈川県川崎市高津区坂戸100番1号 KSPR&Dビ
ネスパークビル 富士ゼロックス株式会社内

㉑ 出 願 人 富士ゼロックス株式
社

東京都港区赤坂3丁目3番5号

㉒ 代 理 人 弁理士 木村 高久

明 細 書

1. 発明の名称

ソースコード生成装置

2. 特許請求の範囲

(1) 複数のグラフィカル・ユーザインターフェース部品がその属性に従って分類された木構造のデータとして表現されるグラフィカル・ユーザインターフェースデータから、特定プログラミング言語のソースコードを生成するソースコード生成装置であって、

前記グラフィカル・ユーザインターフェースデータを、その構成部品の種類、構成部品の属性名、及び構成部品の属性値からなるトークンの列に分解する字句解析手段と、

この分解されたトークンの列に基づき、部品の識別子、部品の名前及び種類、リソースリスト、及びオブジェクトプログラムで使用する変数名によって構成されるデータテーブルを生成する構文

解析手段と、

前記生成対象とするソースコードとして直接に付加される文字列、及び同生成対象とするソースコードとするために前記生成されるデータテーブルに対して補う必要のある情報が記述されたテンプレートと、

前記生成されるデータテーブルを、前記テンプレートに記述されている情報に基づき変換し、かつ、同テンプレートにより指示される部位に埋め込みつつ、対象となるソースコードを生成するコード生成手段と、

を具えるソースコード生成装置。

(2) 前記構文解析手段は、前記部品の名前と前記部品の識別子とをセパレータにて結合して前記オブジェクトプログラムで使用する変数名を生成する

請求項(1)記載のソースコード生成装置。

(3) 前記コード生成手段は、

前記生成対象とするソースコードの各々異なる機能部分について各別にコード生成を行う複数の

特開平4-163625 (2)

サブコード生成手段と、

これら各サブコード生成手段にて生成される各別のコードを、対象とするプログラミング言語の仕様に依じて結合するコードマージ手段と、

を具えて構成され、

前記テンプレートは、前記サブコード生成手段の別に複数用意される

請求項(1)記載のソースコード生成装置。

3. 発明の詳細な説明

〔産業上の利用分野〕

この発明は、ワークステーションやパーソナルコンピュータ等を通じて設計、編集されたグラフィカル・ユーザインターフェースデータから特定プログラミング言語のソースコードを生成するに好適なソースコード生成装置に関する。

〔従来の技術〕

ワークステーションやパーソナルコンピュータ等に視覚的な操作環境を提供するグラフィカル・ユーザインターフェース(以下、GUIと略称す

ols (リポート・オン・ダイアログ・スペシフィケーション・ツールズ)」M.Green(エム・グリーン)著

◆「Human-Computer Interface Development: Concepts and Systems for Its Management (フューマン・コンピュータ・インターフェース・デベロップメント: コンセプト・アンド・システムズ・フォー・イツ・マネージメント)」

H.REX HARTSON (エイチ・レックス・ハートソン) DEBORAH HIX (デボア・ヒックス) 共著

ACM Computing Surveys Vol.21 No.1 1989.3 P.5-92

等々は何れも、GUIについて考察された文献であり、この他にも、様々な形で、GUIに関する研究や開発が進められている。

ところで、このようなGUIを実現するためのオブジェクトプログラムは通常、

(A) GUI部品が所定に組み合わされたデータとして構成されるGUIデータをGUI編集装置を通じて設計、編集する。

る)は、これらワークステーションやパーソナルコンピュータ等を通じての知的生産を著しく高め得る手段或いは技術として、近年、大いに注目を集めている。

例えば、

◆「GMW (ギブ・ミー・モア・ウィンドウズ) ウィンドウ・システム上のアプリケーション構築について」大谷浩司 他著 コンピュータソフトウェア

Vol.17 No.1 (1990.1.16) P.45-60

◆「ヒューマンインタフェースの最先端」日本ソフトウェア科学会 平成2年1月18日発行

P.29-48 「グラフィカルなユーザ・インターフェースとその開発環境について」萩谷昌己著

◆「User Interface Management Systems (ユーザ・インターフェース・マネージメント・システムズ)」G.E.Pfaff (ジー・イー・ファフ) 編 Springer-Verlag (スプリングァー出版)

1983.11.1-3

P.9-20「Report on Dialogue Specification To-

(B) この設計、編集されたGUIデータをソースコード生成装置に読み込んで、特定のプログラミング言語のソースコードを生成する。

(C) この生成されたソースコードを、上記特定のプログラミング言語用のコンパイラによってコンパイルする。

といった手順を経て生成されるが、上記のGUI編集装置やソースコード生成装置は元来、OS (オペレーティングシステム) やプログラミング言語、ライブラリ等々のGUIを構築する環境に大きく依存しているのが普通であることから、特にソースコード生成装置に限っていても、

(1) 上記GUIを構築する環境が異なれば、ソースコードに埋め込まれるテキストやその形式、順序等も異なってくることから、これらGUIの構築環境毎にソースコード生成装置を用意する必要がある。因みに従来は、ソースコードの出力フォーマットが同装置のソースコード生成プログラムに内蔵されているのが普通である。

(2) 上記オブジェクトプログラム内で使用する変

特開平4-163625 (3)

数名もこのソースコード生成装置を通じて自動的に生成されるが、生成されたソースコードを後に人為的に編集しようとした場合、この変数名がどのような目的で使用されているか理解しにくい。すなわち、こうしたソースコード生成装置では、人間が使用しないような文字列の組合せを選んで上記の変数名を自動生成することから、変数名の重複はないものの、その可読性は著しく低いものとなっている。

等々、実用上は、また汎用性を考える場合には、尚多くの問題を抱えている。

なお、上記ソースコード生成装置やコンパイラの概念については、例えば

◆「コンパイラの理論と実現」疋田、石畑 共著
共立出版 1989年 6月
に詳しい。

〔発明が解決しようとする課題〕

上述のように、従来のGUI開発環境におけるソースコード生成装置あっては、上記(1)～(2)として挙げたような課題を残すものであり、特に

よって構成されるデータテーブルを生成する構文解析手段。

(c) 前記生成対象とするソースコードとして直接に付加される文字列、及び同生成対象とするソースコードとするために前記生成されるデータテーブルに対して補う必要のある情報が記述されたテンプレート。

(d) 前記生成されるデータテーブルを、前記テンプレートに記述されている情報に基づき変換しかつ、同テンプレートにより指示される部位に埋め込みつつ、対象となるソースコードを生成するコード生成手段。

を少なくとも具備してソースコード生成装置を構成するようにする。

〔作用〕

上記(c)の態様で必要文字列(テキスト)及び必要情報が記述されたテンプレートを用意することで、更にはまた、上記(d)の態様でこのテンプレートを利用するコード生成手段を具備することで、上記字句解析手段及び構文解析手段を通して生成

現在開発中のものも含む多くのGUI開発環境に対応すべく、その汎用性、並びに実用性を考える上では、これら課題の早急な解決が望まれる。

この発明は、こうした実情に鑑みてなされたものであり、いかなるGUI開発環境にあっても、上記の課題を解決して、汎用性に富み、可読性の高いソースコードを生成することのできるソースコード生成装置を提供することを目的とする。

〔課題を解決するための手段〕

こうした目的を達成するために、この発明では、複数のGUI部品がその属性に従って分類された木構造のデータとして表現されるGUIデータから、特定プログラミング言語のソースコードを生成することを前提に、

(a) 前記GUIデータを、その構成部品の種類、構成部品の属性名、及び構成部品の属性値からなるトークンの列に分解する字句解析手段。

(b) この分解されたトークンの列に基づき、部品の識別子、部品の名前及び種類、リソースリスト、及びオブジェクトプログラムで使用する変数名に

されるデータテーブルであれば、GUI構築環境がどのようなものであれ、その生成されるソースコードは、対象となるプログラミング言語の仕様に応じた所定の規則に従ったものとなる。すなわち、GUI構築環境の違いに基づくソースコードの出力フォーマットの違いといったようなものは上記テンプレートによって吸収されるようになる。したがって、各GUIの構築環境毎に、それぞれその仕様に応じた各別のテンプレートを用意しておき、利用するGUI構築環境に応じて、それに見合ったテンプレートをその都度使い分けるようにすれば、ソースコード生成装置自体は只1つのもので済むようになる。

なおこれに併せ、上記構文解析手段において、部品の名前と部品の識別子とをセパレータ(例えば" _ ")にて結合して前記の変数名を生成するようにすれば、この生成される変数名を、非常に可読性の高いものとして行うことができるようになる。これによって、上記自動生成されるソースコードの後の編集やメンテナンスも容易となる。

特開平4-163625 (4)

また更に、上記コード生成手段を、
(d-1) 生成対象とするソースコードの各々異なる機能部分について各別にコード生成を行う複数のサブコード生成手段。

(d-2) これら各サブコード生成手段にて生成される各別のコードを、対象とするプログラミング言語の仕様に依じて結合するコードマージ手段。
を具えて構成し、上記テンプレートも、これらサブコード生成手段の別に複数用意するようにすれば、ソースコードの行の並びに対して独立にコードを生成することができるようになり、該ソースコード生成装置としての汎用性も更に高いものとなる。

〔実施例〕

第1図に、この発明にかかるソースコード生成装置の一実施例をGUI (グラフィカル・ユーザインターフェース) 編集装置とともに示す。

この実施例では便宜上、GUIの部品として、汎用OSであるUNIX上でのGUI構築環境の1つとしてしられているXウィンドウシステム・

ウシシステム)との間でのメッセージ通信を通じて実行されるようになる。第2図に、このGUI1による表示装置を通じての表示制御画面の一例を示す。

図ろにこの第2図に示す画面において、11は、GUI部品(widget)表示用のウィンドウであるブラウザエリア、12は、GUIデータの編集(設計)用のウィンドウである編集用エリア、13a、13b及び13cは、各々「edit (エディット)」メニュー、「file (ファイル)」メニュー、及び「main (メイン)」メニューをプルダウン表示させるためのメニューボタン、14は、上記ブラウザエリア11に所望のGUI部品を表示させるためのブラウジングボタン、をそれぞれ示している。後に述べるGUIデータの編集(設計)操作等は、全てこのGUI画面を通じて実行されるようになる。

また、第1図において、上記GUI1を通じてユーザ入力によるイベントが通知される編集装置2は、大きくは、該通知されるイベントを都度の

バージョン11の、Athena Widget (アテナ・ウィジェット)を使用し、また生成の対象とするソースコードとしては、C言語で記述されたソースプログラムとする場合について説明する。

はじめに、この実施例のソースコード生成装置に読み込まれるGUIデータの形態について、その大本となる編集装置との兼ね合いのもとに説明する。

まず、第1図において、GUI1は、表示装置(図示せず)の制御プログラムとして、この表示装置上にAthena Widgetで定義されたGUI編集用の画面を視覚的に表示制御するXウィンドウシステムからなる。そしてこのGUI1では、これら表示情報に対するキーボードやマウスなどの入力装置(図示せず)を通じてユーザ入力があった場合、これをイベントとして受け取って、そのイベント内容を、編集装置2にメッセージとして通知する。編集装置2では、以下に説明する全ての処理が、このGUI1(Xウィンド

編集状態に応じて解析する部分である編集動作解析部21と、上記ブラウザエリア11を通じて操作されるGUI部品の管理を主に行う部分であるブラウザ22と、上記編集用エリア12を通じて設計或いは編集されるGUIデータを管理する部分である編集データ管理部23とを具えて構成される。

すなわちこの編集装置2では、例えば

- (1) ブラウザ22によって管理されているGUI部品が上記ブラウザエリア11を通じて指定される。
- (2) この指定された部品が該ブラウザエリア11から上記編集用エリア12の所定位置にマウスによりドラッグされるなどして配置される。
- といったようなユーザによる編集操作に対し、
- (3) 編集動作解析部21が、この間のイベントを解析してその編集内容が「部品の追加」である旨判断し、該編集動作解析部21から上記編集データ管理部23に対して当該GUI部品の追加指令を発する。

特開平4-163625 (5)

(4) これを受けた編集データ管理部23が、その時点で管理している当該編集GUIデータにこの部品の情報を加えて、これを新たなGUIデータとして管理する。

といったかたちで、ユーザによるGUI編集作業を裏面から支えるよう動作する。

また第1図において、記憶装置3は、1つには、上記編集装置2のブラウザ22によって管理されるGUI部品をはじめ、同編集装置2を通じてこうして編集(設計)、管理されるGUIデータを格納、保持するための装置である。この記憶装置3はまた、この実施例のソースコード生成装置における各種作業データの保管装置としても併せ用いられるが、その詳細については、ソースコード生成装置とともに後に詳述する。

ところで、上記編集装置2では、GUIを構成する各部品をオブジェクトとして捉えている。このため、これら部品をクラスという概念で分類することができるようになる。ここでいうクラスとは、同じ特徴を持つオブジェクトの仕様を定義し

たものである。

またここでは、オブジェクト指向における継承(inheritance)の考えを取り入れている。これにより、上記クラスの定義を行うにも、その上位クラスとの差分のみを記述することで足りるようになる。

特に、この編集装置2として、上記のように、編集用エリア12(第2図)に呼び出された各GUI部品及びその編集内容をGUIデータとして管理する部分である編集データ管理部23では、これら各GUI部品をこうしたオブジェクトとして捉えることにより、同部品をそれら各々の持つ属性に従ってカテゴリに分類し、木構造を用いて管理している。ここでのカテゴリは、上記クラスに対応し、下位のカテゴリの定義には、その上位のカテゴリとの差分のみが用いられる。

また、同編集データ管理部23は、上記編集用エリア12に呼び出された部品に対して、その各々に個別の識別子を設定する機能も併せ持つ。これら設定された識別子の値は、その部度、編集動

作解析部21に返されるとともに、該編集データ管理部23によって管理されるGUIデータ自身にも、それを構成する部品の識別情報として付加されるようになる。

第3図に、こうした編集装置2を通じて編集(設計)されるGUIデータの一例を構造図として示す。

この第3図に例示するデータは、先の第2図においてその編集用エリア12内に参考までに付記した部品をイメージしたものであって、例えば、識別子「6522」が付された「Window(ウィンドウ)」部品WWの上に置かれた

- ◇x座標=100
- ◇y座標=200
- ◇ラベル表示="command"
- ◇callback関数名=foo
- ◇識別子=6524

なる「Command(コマンド)」部品CWを想定している。

そしてこの部品CWは、同第3図に示されるよ

うに、1つのオブジェクトとして、大きくは「Property(属性)」、「Identifier(識別)」、及び「Struct(構造)」の各要素からなり、更にこれら各要素のうちの、「Property」要素が、メジャーなカテゴリ、すなわち「Major Props(メジャー属性)」として、上記位置の属性「x=100」及び「y=200」を含む「Geometry(幾何)」情報と、マイナーなカテゴリ、すなわち「Minor Props(マイナー属性)」として、上記文字列及びcallback関数名の属性「label(ラベル表示)=Command」及び「callback(callback関数名)=foo」を含む情報と、からなり、「Identifier」要素が、当該部品CWの識別情報である「id(識別子)=6524」、「name(部品の名前)=command」及び「class(クラス名)=Command」を含む情報からなり、「Struct」要素が、同部品CWの置かれるいわば親部品としての上記

特開平4-163625 (6)

「Window」部品WWの識別子「parent (親部品) = 6522」を含む情報からなる階層構造によって表現されている。

図みに、「Property」要素において、上記位置の属性は全ての部品が持つものの、上記部品上に表示される文字列の属性等は全ての部品が持つとは限らないことから、すなわち、上記の例のような「コマンドボタン」の場合には、こうした文字列の属性を持つが、「スクロールバー」のような部品の場合には、こうした文字列の属性は持たないことから、上記位置の属性は、全ての部品に通用するメジャーなカテゴリとしてカテゴリ「major Props」に、他方の文字列等の属性は、全ての部品には通用しないマイナーなカテゴリとしてカテゴリ「minor Props」に、それぞれ分類されている。

このように、上記編集装置2を通じて編集され、管理されるGUIデータの情報は、第3図に示されるように、アイテムとそのアイテムの組によって記述される。そして上述のように、各アイテム

は、類似した性質のもの同士がカテゴリにまとめられ、更に各カテゴリも、類似した種類のもの同士が上位のカテゴリにまとめられる階層構造となっている。この実施例においては、これをS式を用いて表記する。

S式とは、人工知能用言語である「LISP (リスプ)」において多く用いられる「Symbolic expression (シンボリック・エクスプレッション)」の略である。このS式では、アイテムの構造を「かっこ()」の組み合わせを用いて表記するものであり、同じ階層にあるアイテムについては、これを同じ組の「かっこ()」の中に記述し、それより1つ下の階層にあるアイテムについては、これを上位の「かっこ()」の中に包含される別の組の「かっこ()」の中に記述することで、その階層構造を表現するようにしている。

例えば、第3図に例示される上記の部品の情報は、該S式によって、

```
(#Object
```

```
) ... (1)
```

のように表記されることとなる。

このように、編集(設計)されるGUIデータを、S式によって汎用的に表記される階層構造を用いて管理し、保存するようにすることで、GUI環境によって限定されない、より汎用性に富んだ情報管理が実現されるようになる。

なお、上記部品CWの置かれるいわば親部品としての上記「Window」部品WWは、同S式によって

```
(#Property
  (#MajorProps
    (#Geometry
      (@x $100)
      (@y $200)
    )
    (#MinorProps
      (@label $Command)
      (@callback $foo)
    )
  )
)
(#Identifier
  (@id $6524)
  (@name $Command)
  (@class $Command)
)
(#Struct
  (@parent $6522)
)
```

```
(#Object
  (#Property
    (#MajorProps
      (#Geometry
        (@x $0)
        (@y $0)
      )
    )
  )
  (#MinorProps
```



```

    ($Geometry
      (@width $300)
      (@height $400)
    )
  )
)
($Identifier
  (@id $522)
  (@name $shell)
  (@class $Shell)
)
($Struct
  (@parent NULL)
)
... (2)

```

のように表記されているとする。

さて、この実施例によるソースコード生成装置は、このようにS式によって表記され、記憶装置

ために上記生成されたデータテーブルに対して補う必要のある情報がそれぞれ記述されているものとする。

以下、これら各部の具体的な機能、並びに動作について、第4図～第10図を併せ参照して順次詳述する。

まず、字句解析部41では上記のように、(1)式、或いは(2)式として表記されるような編集データを記憶装置3から読み込んで、これをトークンの列に分解する。ここで分解されるトークンは、「LBRANCH (左かっこ)」、「RBRANCH (右かっこ)」、「KEY (構造を示す名前)」、「LITERAL (オブジェクトの属性値)」、及び「IDENT (オブジェクトの属性名)」を単位としたものである。

因みに上記(1)式或いは(2)式において、「#」で始まる文字列は「KEY (構造を示す名前)」であり、「@」で始まる文字列は「IDENT (オブジェクトの属性名)」であり、「\$」で始まる文字列は「LITERAL (オブジェク

特開平4-163625 (7)

3上に保存されるGUI編集データを読み込んで、これからC言語で記述されたソースプログラムを生成するものである。

第1図に併せ示すように、このソースコード生成装置は、上記読み込んだデータを、部品の属性名、部品の属性値、及び部品のクラスからなるトークンの列に分解する字句解析部41と、この分解されたトークンの列から、部品の識別子、部品のクラス名、部品の名前、部品の属性(リソースリスト)、及び生成目的とするプログラムで使用する変数名により構成されるデータテーブルを生成する構文解析部42と、この生成されたデータテーブルを、上記記憶装置3上に予め用意されたテンプレート31に記述されている情報に基づき変換し、かつ、同テンプレート31により指示される部位に埋め込みつつ、対象となるソースコードを生成するコード生成手段と、を具えて構成される。テンプレート31には、生成対象とするソースコードとして直接に付加される文字列(テキスト)、及び同生成対象とするソースコードとする

トの属性値)」である。したがって、第4図に(A)として示す上記(1)式にて表記される部品の場合には、この字句解析部41による上記処理によって、同第4図(B)に示されるように、「(」、「#Object」、「(」、「#Property」、「(」、「#MajorProps」、「(」、「#Geometry」、「(」、「@x」、「\$100)」、「(」、「@y」、「\$200)」、「)」、「(」、「#MinarProps」、「(」、「@label」、「\$Command)」、「(」、「@callback」、「\$foo)」、「)」、「)」、「(」、「#Identifier」、「(」、「@id」、「\$6524)」、「(」、「@name」、「\$command)」、「(」、「@class」、「\$Command)」、「)」、「(」、「#Struct」、「(」、「@parent」、「\$6522)」、「)」、「)」及び「)」といったトークンの列に分解されることとなる。こうして分解されたトークンの

特開平4-163625 (8)

列は配列に格納される。

こうして、字句解析部41による分解を終えると、次の構文解析部42では、これらトークンが格納されている配列を受け取って、上記のように、部品の識別子、部品のクラス名、部品の名前、リソースリスト、及びプログラムの中で使われる変数名、の各フィールドを持つ第4図(C)に示されるようなデータテーブルを記憶装置3上に生成する。

ここで、該構文解析部42による上記変数名の生成手順について、第5図を併せ参照しつつ説明する。

先にも述べたように、従来のソースコード生成装置においても、こうした変数名を自動生成する機能はあったが、これらは何れも、通常人間が使用しないような文字列の組合せを選んで変数名とすることから、確かに変数名の重複はないものの、その可読性は著しく低いものとなっていた。

そこでこの実施例のソースコード生成装置では、上記部品の名前とGUIデータ編集時に設けられ

格納する。第4図に示す例においては、値「command」が取り出され、これが「部品の名前」のフィールドに格納される。

といった一連の処理の後、同データテーブルの「変数名」のフィールドに格納すべく値を生成するために、同第4図に示す例においては、上記属性の名前「name」の値である「command」と、上記属性の名前「id」の値である「6524」とを、該構文解析部42内に定義されている変数名生成関数に渡す。

これにより変数名生成関数では、

(4) convertedID=Convert(id)を呼び、上記識別子「id」の値「6524」を文字列に変換してこれを変数convertedIDに代入する(第5図ステップS11)。

(5) 次に、strcat(widgetName, " _")を呼び、変数widgetNameに上記部品の名前「name」の値「command」に応じた「command _」を代入する。すなわち、こうした文字列結合関数によって、「部品の名前」の後にセパレータ「_」

た上記識別子とを有効利用して、構文解析部42を通じた以下に列記する手順によって、可読性の高い変数名を自動生成するようにしている。

すなわち構文解析部42では、

(1) 上記配列を受け取るとまず、部品の識別子を決定するために、属性の名前「id」の値を取り出し、これを当該部品の識別子として上記データテーブルの「識別子」のフィールドに格納する。第4図に示す例においては、値「6524」が取り出され、これが「識別子」のフィールドに格納される。

(2) 次に、同配列から属性の名前「class」の値を取り出し、これを当該部品のクラス名として上記データテーブルの「クラス名」のフィールドに格納する。第4図に示す例においては、値「Command」が取り出され、これが「クラス名」のフィールドに格納される。

(3) 同様に、配列から属性の名前「name」の値を取り出し、これを当該部品の名前として上記データテーブルの「部品の名前」のフィールドに

が付加されるかたちとなる(第5図ステップS12)。

(6) 最後に、文字列結合関数strcat(widgetName, convertedID)を呼び、これら代入した文字列「command _」及び「6524」を結合して、変数名「command _6524」を生成する(第5図ステップS13)。

といった手順にて変数名を生成し、この生成した変数名「command _6524」を構文解析部42に返す。

構文解析部42では、こうして返された変数名「command _6524」を、上記生成するデータテーブルの「変数名」のフィールドに加えて(格納して)、その処理を終える。

このように、この実施例によれば、部品の名前と部品の識別子とをセパレータにて結合してオブジェクトプログラムで使用する変数名を生成するようにしていることから、自動生成されたソースコードを後に人為的に編集する場合でも、そこで使われている変数名の用途等については、容易に

特開平4-163625 (9)

理解することができるようになる。また、変数名の一部として、部品毎に各別の値として付される識別子を用いていることから、同一の変数名が重複して生成されることもない。

こうして構文解析部42を通じてデータテーブルが生成されると、コード生成部43では、記憶装置3上に用意されている上記テンプレート31を用いて、この生成されたデータテーブルに応じたC言語ソースコードを生成する。

コード生成部43は、第6図に示すように、callbackサブコード生成部431、resourceサブコード生成部432、includeサブコード生成部433、widget_varサブコード生成部434、及びwidget_createサブコード生成部435の5つのサブコード生成部と、これらサブコード生成部431~435を通じて生成されるサブコードを所定に結合して、上記所望とされるソースコードを最終的に出力するコードマージ部430と、を具えて構成される。

れている情報に基づき、同データテーブルから、widget(ウィジェット:部品)変数の宣言のプログラムのみについてそのソースコードを生成する部分であり、そしてwidget_createサブコード生成部435は、同テンプレート31のうちの第5テンプレート31eに記述されている情報に基づき、同データテーブルから、部品の生成に関するプログラムのみについてそのソースコードを生成する部分である。これら各サブコード生成部によるソースコード生成手順を第7図に示す。

すなわち、上記第1~第5テンプレート31a~31eは、前述のように、各々割り当てられたプログラムについて、そのソースプログラムに直接出力される情報とこれに補充する必要がある情報とが所定の形式で記述されたファイルであり、これらサブコード生成部431~435では何れも、各々対象となるテンプレートへの記述情報に基づき、また、上記生成されたデータテーブルに基づき、以下に列記する手順をもって、その割り

callbackサブコード生成部431は、記憶装置3上に用意されたテンプレート31のうちの第1テンプレート31aに記述されている情報に基づき、上記構文解析部24にて生成されたデータテーブル(第4図(C)参照)から、callback(コールバック)関数のプログラムのみについてそのソースコードを生成する部分であり、resourceサブコード生成部432は、同テンプレート31のうちの第2テンプレート31bに記述されている情報に基づき、同データテーブルから、resource(リソース)のプログラムのみについてそのソースコードを生成する部分であり、includeサブコード生成部433は、同テンプレート31のうちの第3テンプレート31cに記述されている情報に基づき、同データテーブルから、include(インクルード)のプログラムのみについてそのソースコードを生成する部分であり、widget_varサブコード生成部434は、同テンプレート31のうちの第4テンプレート31dに記述さ

当てられたソースコード(サブコード)を生成する。

- (1) まず、対象となるテンプレートを読み込む(第7図ステップS21)。
- (2) この読み込んだテンプレートに記述されている情報を解釈して、補充しなければならない情報を生成するのに必要なデータをリストアップする(第7図ステップS22)。
- (3) 上記データテーブルから該当するデータを取り出す(第7図ステップS23)。
- (4) この取り出したデータを、上記テンプレートに記述されている情報に基づき、そこで指定されている形式の情報に変換する(第7図ステップS24)。
- (5) この変換した情報を、当該テンプレートの指定されている部分に埋め込む(第7図ステップS25)。

第8図は、上記各テンプレート31a~31eについてその具体例を例示したものであり、また

特開平4-163625 (10)

第9図は、これらテンプレート31a~31eに基づく上記各サブコード生成部431~435のソースコード(サブコード)生成出力例を示したものであり、以下、これら第8図及び第9図を対比しつつ、上記各サブコード生成部によるソースコード生成処理を更に具体的に説明する。なお、この例においては、上記テンプレートに記述される情報のうち、補充しなければならない情報についてはこれを、文字列「<\$」と文字列「\$>」とによって囲んで定義するものとする。

いま、構文解析部42によって生成された上記データテーブルがコード生成部43に渡されたとすると、callbackサブコード生成部431は、第8図(a)に示される内容を有する第1テンプレート31aを読み込んで、コールバック関数のプログラムについてそのソースコードの生成を開始する。

この際、該callbackサブコード生成部431では、上記データテーブル(第4図(C)参照)の「リソースリスト」フィールドから属性

の名前「x」とその属性値「100」、属性の名前「y」とその属性値「200」、属性の名前「label」とその属性値「Command」、及び同部品の識別名「command」をそれぞれ取り出し、上記第2テンプレート31bの

```
<$ identifier _name $>
```

の部分に、識別名「command」を、

```
<$ resource _name $>
```

の部分に、上記各属性の名前「x」、「y」、「label」を、また

```
<$ resource _value $>
```

の部分に、上記各属性の値「100」、「200」、「Command」をそれぞれ埋め込む。そして、この第2テンプレート31bにこれら各要素

の名前「callback」の属性値「foo」を取り出し、上記第1テンプレート31aのコールバック関数埋め込み指定部分である

```
<$ func _name $>
```

の部分に、この取り出した「foo」を埋め込む。そして、この第1テンプレート31aに「foo」を埋め込み生成したコードを、「callbackサブコード」としてコードマージ部430に出力する(第9図(a)参照)。

また、同データテーブルがコード生成部43に渡されると、resourceサブコード生成部432は、第8図(b)に示される内容を有する第2テンプレート31bを読み込んで、リソースのプログラムについてそのソースコードの生成を開始する。

この際、該resourceサブコード生成部432では、上記データテーブル(第4図(C)参照)の「リソースリスト」フィールドから属性

を埋め込み生成したコードを、「resourceサブコード」としてコードマージ部430に出力する(第9図(b)参照)。

同じく、コード生成部43にデータテーブルが渡されると、includeサブコード生成部433は、第8図(c)に示される内容を有する第3テンプレート31cを読み込んで、インクルードプログラムについてそのソースコードの生成を開始する。

この際、該includeサブコード生成部433では、同データテーブル(第4図(C)参照)の「クラス名」フィールドから当該部品のクラス名「Command」を取り出し、上記第3テンプレート31cの

```
<$ Header Widget_Class $>
```

の部分に、この取り出したクラス名「Command」を埋め込む。そして、この第3テンプレート31cに「Command」を埋め込み生成し

特開平4-163625 (11)

たコードを、「includeサブコード」としてコードマージ部430に出力する(第9図(c)参照)。

同じく、widget_varサブコード生成部434は、データテーブルがコード生成部43に渡されると、第8図(d)に示される内容を有する第4テンプレート31dを読み込んで、widget変数の宣言のプログラムについてそのソースコードの生成を開始する。

この際、該widget_varサブコード生成部434では、同データテーブル(第4図(C)参照)の「変数名」フィールドから部品を保持する変数名「command_6524」を取り出し、上記第4テンプレート31dの

```
<$ variable __name $>
```

の部分に、この取り出した変数名「command_6524」を埋め込む。そして、この第4テンプレート31dに「command_6524」

する親部品のオブジェクトから第4図に示した例と同様に生成されるデータテーブル(図示せず)の「変数名」のフィールドから親部品の変数名「shell_6522」を取り出し、これら取り出した各データ要素を上記第5テンプレート31eの各々指定される部分に埋め込む。なお、第8図(e)に示すこの第5テンプレート31eにおいて、文字列「\$\$」で囲まれた部分は条件式であって、この場合、「class」が「shell」である場合には、前半の

```
XtAppContext app_con;
<$ variable __name $> = XtAppInitialize(
&app_con, "<$ widget __name $>".NULL, ZERO,
&argc, argv, fallback_resources, NULL, ZERO);
```

と記述された部分を使い、それ以外の場合には、後半の

を埋め込み生成したコードを、「widget_varサブコード」としてコードマージ部430に出力する(第9図(d)参照)。

そして、widget_createサブコード生成部435は、データテーブルがコード生成部43に渡されると、第8図(e)に示される内容を有する第5テンプレート31eを読み込んで、部品の生成のプログラムについてそのソースコードの生成を開始する。

この際、該widget_createサブコード生成部435では、同データテーブル(第4図(C)参照)の「クラス名」のフィールドから当該部品のクラス名「Command」を、「部品の名前」のフィールドから当該部品の名前「command」を、「変数名」フィールドから当該部品の変数名「command_6524」をそれぞれ取り出すとともに、同データテーブルの「リソースリスト」のフィールドの属性名「parent」の属性値によって案内されるデータテーブル、すなわち前記(2)式にて表記されると

```
<$ variable __name $> = XtCreateManagedWidget("<$ widget __name $>". <$ widget __class $>WidgetClass, <$ parent__variable $>, NULL, ZERO);
```

と記述された部分を使うことを意味する。ここでの例では、「class」が「Command」となっていることから、後者の部分が使われる。したがってこの例の場合、widget_createサブコード生成部435では、該第5テンプレート31eにおける後半部分の

```
<$ variable __name $>
```

の部分に、当該部品の変数名「command_6524」を、

```
<$ widget __name $>
```

の部分に、当該部品の名前「command」を、

特開平4-163625 (12)

また

```
<$ widget _class $>
```

の部分に、当該部品のクラス名「Command」を、そして

```
<$ parent _variable $>
```

の部分に、親部品の変数名「shell_6522」をそれぞれ埋め込む。そして、この第5テンプレート31eにこれら各要素を埋め込み生成したコードを、「widget_createサブコード」としてコードマージ部430に出力する(第9図(e)参照)。

これら各サブコード生成部による以上のサブコード生成処理は、前記(2)式にて表記されるとした親部品のオブジェクトに対しても同様に実施される。

これらの各サブコードが入力されるコードマ

ージ部434によって生成された第9図(d)に示されるwidget_varサブコードと、widget_createサブコード生成部435によって生成された第9図(e)に示されるwidget_createサブコードとをマージするとともに、C言語の予約語である

```
main(argc, argv)
int argc;
char **argv;
{
    ...
}
```

を追加して、第10図(b)に示されるような部品(Widget)を生成するプログラムのソースコードを生成する。

(4) この生成したソースプログラムを記憶装置3に出力する。

ジ部430は、上述のように、これらサブコードを所定に結合して、上記所望とされるソースコードを出力する部分であり、具体的には、以下の手順によって、第10図に示されるようなC言語のソースプログラムを最終的に生成し、この生成したコードを記憶装置3に出力する。

(1) includeサブコード生成部433によって生成された第9図(c)に示されるincludeサブコードと、callbackサブコード生成部431によって生成された第9図(a)に示されるcallbackサブコードとをマージして、第10図(a)に示されるようなコールバック関数のソースプログラムを生成する。

(2) この生成したソースプログラムを記憶装置3に出力する。

(3) includeサブコード生成部433によって生成された第9図(c)に示されるincludeサブコードと、resourceサブコード生成部432によって生成された第9図(b)に示されるresourceサブコードと、wi

doget_varサブコード生成部434によって生成された第9図(d)に示されるwidget_varサブコードと、widget_createサブコード生成部435によって生成された第9図(e)に示されるwidget_createサブコードとをマージするとともに、C言語の予約語である

```
Widget shell_6522;
```

は、上記親部品のオブジェクトに関してwidget_varサブコード生成部434が生成したwidget_varサブコードの部分であり、また

```
XtAppContext app_con;
```

```
shell_6522 = XtAppInitialize(&app_con,
    "shell", NULL, ZERO, &argc, argv, fallback_re
sources, NULL, ZERO);
```

は、同じく上記親部品のオブジェクトに関してwidget_createサブコード生成部434が生成したwidget_createサブコ

特開平4-163625 (13)

ードの部分である。

このように、この実施例によるソースコード生成装置では、上記コード生成部43を、生成対象とするソースコードの各々異なる機能部分について各別にコード生成を行う複数の(上記の例では5つの)サブコード生成部と、これら各サブコード生成部にて生成される各別のコードを、対象とするプログラミング言語の仕様に依じて結合するコードマージ部とを具備して構成するとともに、上記テンプレート31も、これらサブコード生成部の別に、各々それに見合ったものを複数用意するようにしたことから、上記のように、ソースコードの行の並びに対して独立に所望のコードを生成して、これを任意に結合することができるようになる。このため、GUI環境や生成の対象となるソースプログラムの言語仕様が異なる場合であっても、具体的には、ソースプログラムに埋め込まれるテキストの順の等が異なる場合であっても、従来のように、これらGUI環境や言語仕様の別にソースコード生成装置を用意する必要はなくな

い。ただしこの場合、その表記方法と字句並びに構文解析方法との対応を新たに定義する必要はある。

【発明の効果】

以上説明したように、この発明によれば、生成対象とするソースコードとして直接に付加される文字列、及び同生成対象とするソースコードとするために構文解析された内容に対して補う必要のある情報が記述されたテンプレートを設け、このテンプレートを通じて、異なるGUI構築環境間における異なるコード生成要素を吸収するようにしたことから、各種のGUI構築環境に対し汎用的に、目的とされるプログラムのソースコードを生成することができるようになる。

4. 図面の簡単な説明

第1図は、この発明にかかるソースコード生成装置の一実施例をGUI編集装置とともに示すブロック図である。

第2図は、GUI編集装置における編集画面の

り、該ソースコード生成装置としての汎用性も著しく高められるようになる。

なお、上記の実施例では便宜上、Xウィンドウシステム・バージョン11の、Athena Widget (アテナ・ウィジェット)を使用し、また生成の対象とするソースコードとしては、C言語で記述されたソースプログラムとする場合について説明したが、この発明にかかるソースコード生成装置が、こうした環境やOS (オペレーティングシステム)、更には言語仕様に限定されるものでないことは勿論である。

また、このソースコード生成装置において読み込み対象とするGUI編集データも、必ずしも前述したS式を用いて表記する必要はない。要は、当該GUIデータをオブジェクトの集合として捉えてこれらを所要に分類でき、その各要素の属性等を的確に表すことのできる表記方法であれば、すなわち、その字句解析並びに構文解析を通じて、第4図(C)に例示した態様に準じた形でのデータテーブル生成を可能とする表記方法であればよ

一例を示す平面図である。

第3図は、上記のGUI編集装置で編集され、上記実施例ソースコード生成装置に読み込まれるGUI部品情報の管理構造について、その一例を模式的に示す略図である。

第4図は、同実施例ソースコード生成装置による字句解析処理及び構文解析処理を通じて生成されるトークンの列及びデータテーブルについて、その一例を模式的に示す略図である。

第5図は、同実施例ソースコード生成装置による変数名生成手順を例示するフローチャートである。

第6図は、同実施例ソースコード生成装置のコード生成部について、その具体構成と記憶装置との係わりを示すブロック図である。

第7図は、上記コード生成部を構成する各サブコード生成部のソースコード(サブコード)生成手順を示すフローチャートである。

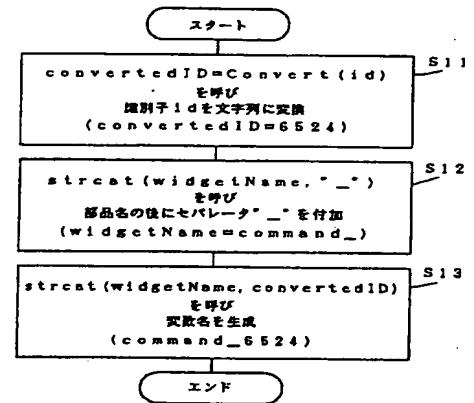
第8図は、上記記憶装置上に用意されるテンプレートの具体例を示す略図である。

特開平4-163625 (14)

第9図は、この第8図に示されるテンプレートに基づき上記各サブコード生成部が生成するサブコード例を示す略図である。

第10図は、この実施例ソースコード生成装置によって最終的に生成出力されるソースコード(ソースプログラム)の一例を示す略図である。

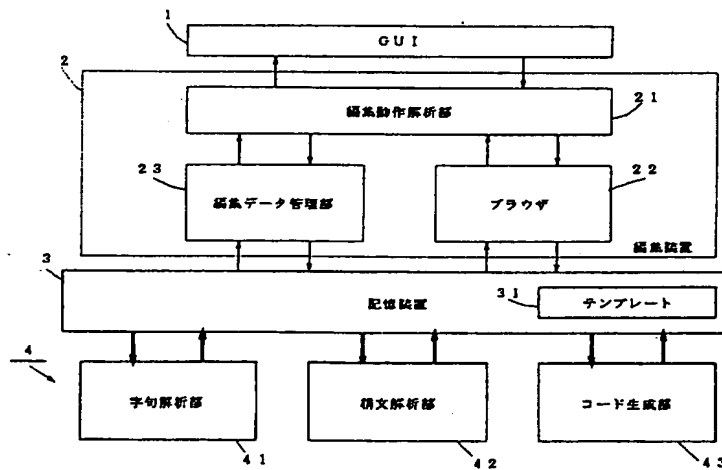
1...GUI、2...編集装置、21...編集動作解析部、22...ブラウザ、23...編集データ管理部、3...記憶装置、31...テンプレート、4...ソースコード生成装置、41...字句解析部、42...構文解析部、43...コード生成部、430...コードマージ部、431~435...サブコード生成部。



出願人代理人 木村 高久

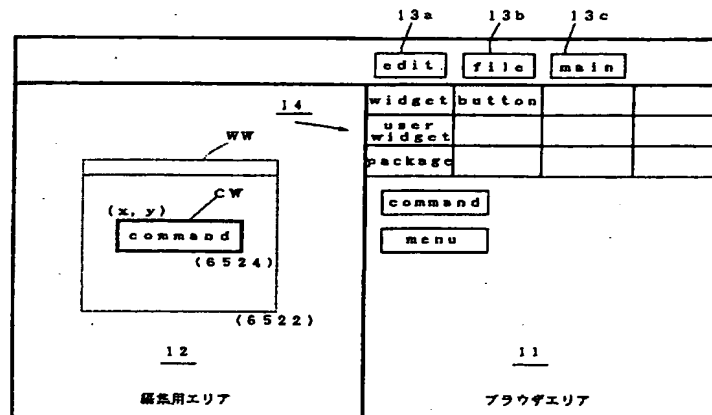


第5図

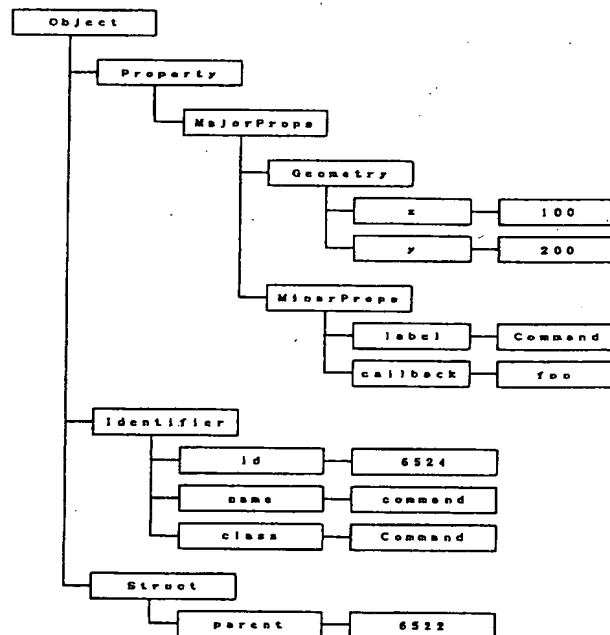


第1図

特開平4-163625 (15)

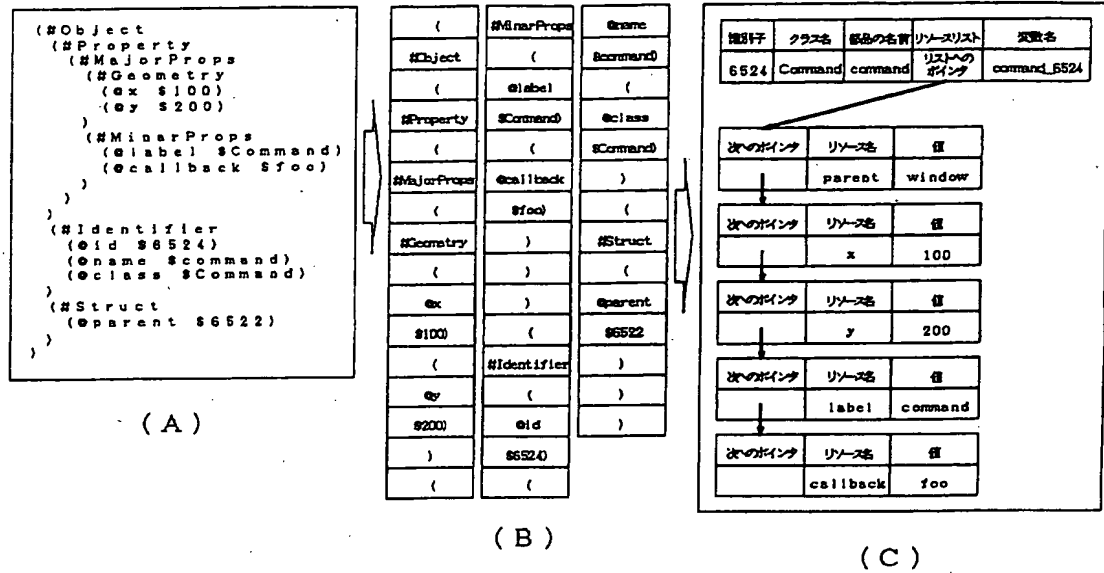


第2図

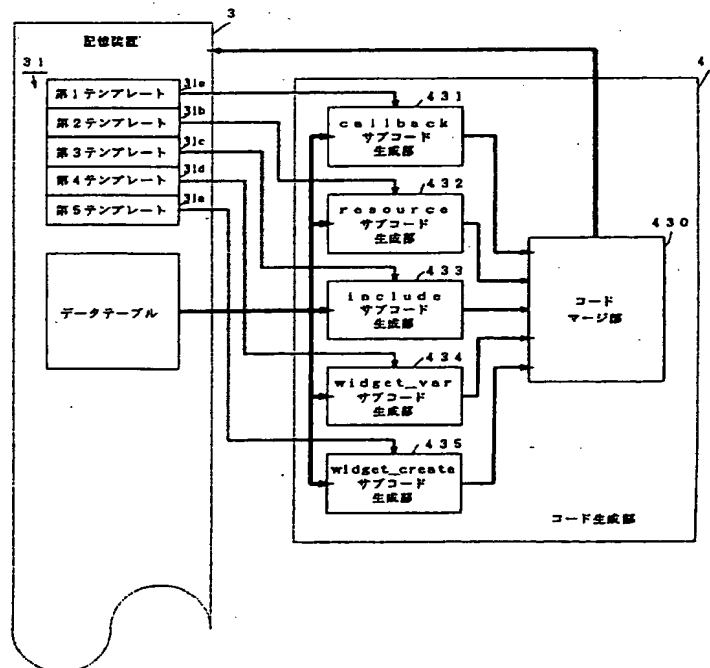


第3図

特開平4-163625 (16)

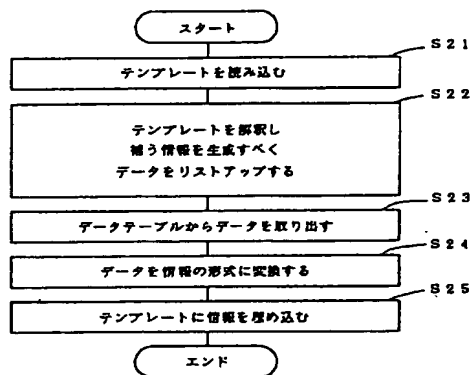


第4図

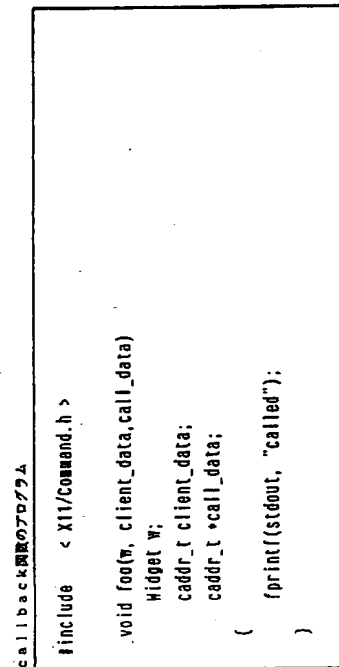


第6図

特開平4-163625 (17)



第7図



第10図(a)

(a) 第1テンプレート31a

```

void <$ func_name $>(w, client_data, event)
{
    Widget w;
    caddr_t client_data;
    caddr_t *call_data;
    {
        printf(stdout, "called");
    }
}
  
```

(b) 第2テンプレート31b

```

String fallback_resources[] = {
    "<$ Identifier_name $>.XtN<$ resource_name $>, <$ resource_value $>",
    NULL
};
  
```

(c) 第3テンプレート31c

```

#include "X11/<$ Header Widget_Class $>.h"
  
```

(d) 第4テンプレート31d

```

Widget <$ variable_name $>;
  
```

(e) 第5テンプレート31e

```

$${IF (class == shell)} $$
    XtAppContext app_con;
    <$ variable_name $> = XtAppInitialize(&app_con, "<$ widget_name $>",
    NULL, ZERO, &argc, argv, fallback_resources, NULL, ZERO);
$${ELSE}
    <$ variable_name $> = XtCreateManagedWidget("<$ widget_name $>",
    <$ widget_class $>WidgetClass, <$ parent_variable $>, NULL, ZERO);
$${ENDIF}
  
```

第8図

特開平4-163625 (18)

(a) callbackサブコード生成出力

```
void foo(w, client_data, event)
Widget w;
caddr_t client_data;
caddr_t call_data;
{
    fprintf(stdout, "called");
}
```

(b) resourceサブコード生成出力

```
String fallback_resources[] = {
    "command.XtNx, 100",
    "command.XtNy, 200",
    "command.Xlabel, Command",
    NULL
};
```

(c) includeサブコード生成出力

```
#include < X11/Command.h >
```

(d) widget_varサブコード生成出力

```
Widget command_6524;
```

(e) widget_createサブコード生成出力

```
command_6524 = XtCreateManagedWidget("command", CommandWidgetClass,
shell_6522, NULL, ZERO);
```

第9図

部品 (Widget) を生成するプログラム

```
#include < X11/Command.h >

String fallback_resources[] = {
    "command.XtNx, 100",
    "command.XtNy, 200",
    "command.Xlabel, Command",
    NULL
};

main(argc, argv)
int argc;
char **argv;
{
    Widget shell_6522;
    Widget command_6524;
    XtAppContext app_con;

    shell_6522 = XtAppInitialize(&app_con, "shell", NULL, ZERO,
    &argc, argv, fallback_resources, NULL, ZERO);
    command_6524 = XtCreateManagedWidget("command", CommandWidgetClass,
    shell_6522, NULL, ZERO);
}
```

第10図 (b)